

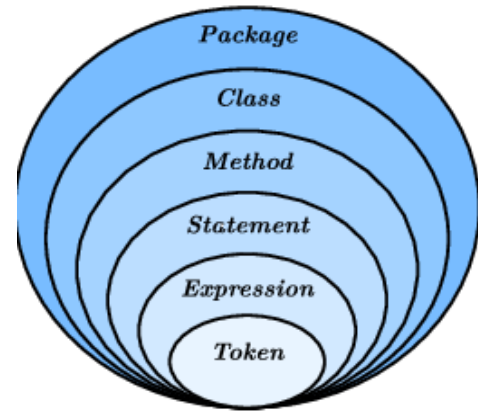
CE142: OBJECT ORIENTED PROGRAMMING WITH C++

February 2023 – May 2023

UNIT – 3

Tokens and Expressions & Control Structure

Objectives



- Learn about tokens used in C++
- Learn about control structures in C++
- Type compatibility
- Variables
- Explore how to form and evaluate expressions in C++
- Cast Operator

Content

- Tokens Keywords, identifiers and constants
- Basic Data Types and user defined data types and derived data types, symbolic constants
- Type compatibility, Declaration of variables, Dynamic initialization,
- Reference variables ,Scope Resolution Operator, Memory Management Operator,
- Manipulators, Type cast operator ,Expressions and their types, implicit Conversion Operator Precedence and Control Structure

Tokens

- The Smallest Units in a program are known as token.
- C++ has following tokens:
 - Keywords
 - Identifiers
 - Constants
 - Strings
 - Operators

Keywords

- Keywords are reserved words in programming .
- Each keyword has fixed meaning and that can not be changed by user.
- For Ex:

`int num1;`

where `int` is keyword indicates `num1` is integer type.

asm	double	new	switch
auto	else	operator	template
break	enum	private	this
case	extern	protected	throw
catch	float	public	try
char	for	register	typedef
class	friend	return	union
const	goto	short	unsigned
continue	if	signed	virtual
default	inline	sizeof	void
delete	int	static	volatile
do	long	struct	while
bool	export	reinterpret_cast	typename
const_cast	false	static_cast	using
dynamic_cast	mutable	true	wchar_t
explicit	namespace	typeid	

Identifiers And Constants

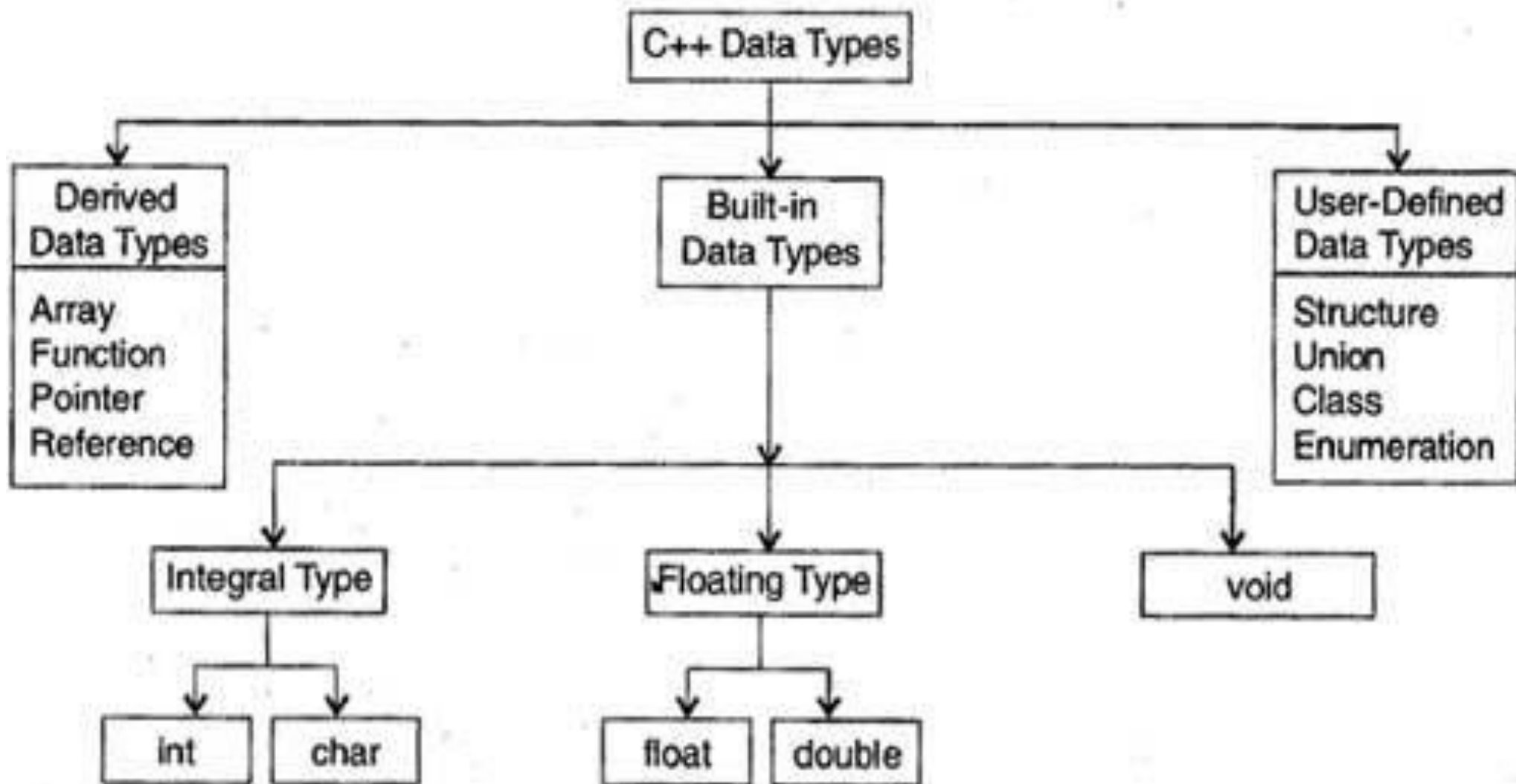
- Identifier is user defined names of variables, functions, arrays, classes, etc.
- They are the fundamental of any language.
- Each language has its own rules for naming these identifiers. These are follow:
 - Only alphabetic characters, digits and underscores are permitted.
 - The name can not start with digits.
 - Uppercase and lowercase letters are distinct.
 - A declared keyword cannot be used as a variable name.

Constants

- Constants refers to fixed values that do not change during the execution of program.
- They include integer, character, floating point numbers and string

C++ Data Types

- C++ data types divided mainly in three categories as follows:
 - Basic data types
 - User defined data types
 - Derived data types



Various Data Types in C++

Basic Data Types

TYPES	BYTES	RANGE
char	1	-128 to 127
unsigned char	1	0 to 255
signed char	1	-128 to 127
int	2	-32768 to 32767
unsigned int	2	0 to 65535
signed int	2	-32768 to 32767
short int	2	-32768 to 32767
unsigned short int	2	0 to 65535
signed short int	2	-32768 to 32767
long int	4	-2147483648 to 2147483647
signed long int	4	-2147483648 to 2147483647
unsigned long int	4	0 to 4294967295
float	8	3.4E-38 to 3.4E_38
double	10	1.7E-308 to 1.7E+308
long double		3.4E-4932 to 1.1E+4932

User-defined Data Types

- There are following user defined data types:
 - Struct
 - Union
 - Enum
 - class

Struct

- A structure is a collection of simple variable
- The variable in a structure can be of different type such as int,char,float,char,etc.
- This is unlike the array which has all variables of same data type.
- The data items in a structure is called the member of structure.
- Syntax:

```
struct structure_name{ declaration of data member;  
};
```
- You can access data member of stucture by using only structure variable with dot operator.

Exercise

Create a structure named student. Enter the student name, marks of C++, maths, Physics and DFS subjects. Calculate the average of the marks and store it in a variable named per. Make per as float variable.

Solution

```
#include<iostream>
using namespace std;

struct
student{
    int Roll_No; char name[15];
    int cPlus,maths,sql,dfs,total;
    float per;
};

int main(){
    struct student s; cout<<"Enter
    Roll No:"; cin>>s.Roll_No;
    cout<<"Enter Name:";
    cin>>s.name;
    cout<<"Enter Marks For CPlus:";
    cin>>s.cPlus;
    cout<<"Enter Marks For Maths:";
```

```
cin>>s.maths;
cout<<"Enter Marks For Sql :";
cin>>s.sql;
cout<<"Enter Marks For DFS:"; cin>>s.dfs;

s.total=s.cPlus+s.maths+s.sql+s.dfs;
s.per=s.total/4;

cout<<"Your Roll No is:"<<s.Roll_No<<"\n";
cout<<"Your Name is:"<<s.name<<"\n";
cout<<"Your Total is:"<<s.total<<"\n";
cout<<"Your Percentage is:"<<s.per<<"%"<<"\n";
return 0;
}
```


C:\Users\Asus\Desktop\C++Programes\STRUCT_pro.exe

```
Enter Roll No:1
Enter Name:meena
Enter Marks For CPlus:89
Enter Marks For Maths:76
Enter Marks For Sql :78
Enter Marks For DFS:86
Your Roll No is:1
Your Name is:meena
Your Total is:329
Your Percentage is:82%
```

```
Process returned 0 (0x0)   execution time : 19.858 s
Press any key to continue.
```

Union

- Union is also like a structure means union is also used to storing different data types ,but the difference between structure and union is that structure consume the memory of addition of all elements of different data types but a union consume the memory that is highest among all elements.
- It consume memory of highest variable and it will share the data among all other variable.

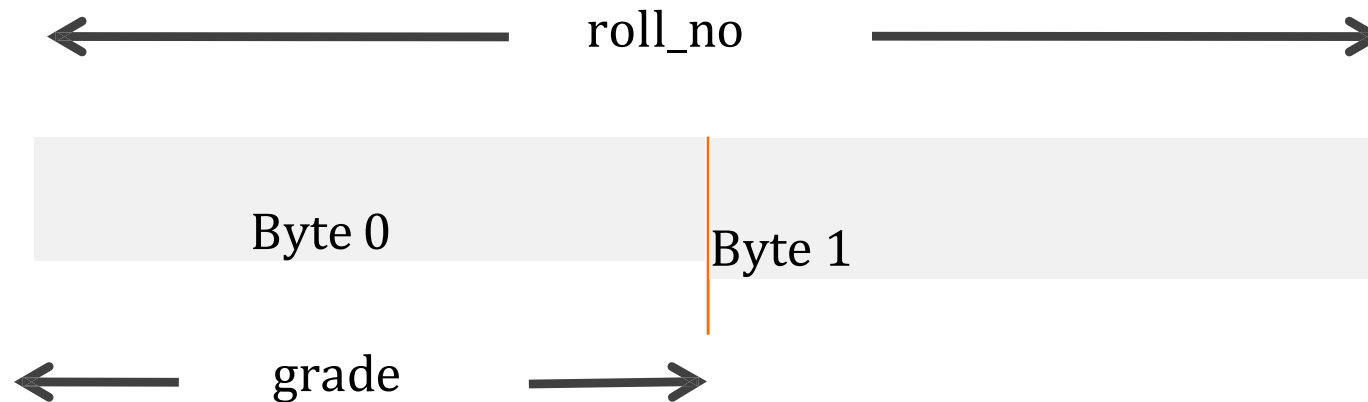
- For example:

If a union contains variable like int, char, float then the memory will be consume of float variable because float is highest among all variable of data type.

- Syntax:

```
union union_name           //union declaration
{
    data_type member1;
    data_type member2;
    ...
};
```

- Like ,
union stu{
 int roll_no; //occupies 2 bytes
 char grade ; //occupies 1 bytes
};



- Memory representation in union

Struct

- Define with '**struct**' keyword.
- The size of object is equal to the sum of individual size of the member object.
- Members are allocated distinct memory location.

Union

- Define with '**union**' keyword.
- The size of object is equal to the size of largest member object.
- Members are share common memory space.

Enum In C++

- Approach to define your own datatype is the enumerated datatype.
- An enumeration type is user defined type that enable the user to define the range of values for the type.
- Enumerated means all the values are listed.
- An enum specifier defines the set of all the names that will be permissible values of the type.
- These permissible values are called members.
- Syntax:

`enum [enum-type] {enum-list};`

- Name constants are used to represent the value of an enumeration for example:

```
enum char{a,b,c};
```

- The default value assigned to the enumeration constant are zero-based so in above example a=0,b=1 and so on.
- The user also can assign value to one or more enumeration constant, and subsequent values that are not assigned will be incremented.

For example :

```
enum char{a=3,b=7,c,d};
```

here, value of c is 8 and d is 9.

Exercise

- Write a program to demonstrate the working of enum. Ask the user to enter the choice of fruits like apple, orange, guava, pineapple (these constants refers to the value of enum) and then print the name of the fruit.

Hint: Use enum & switch case

Solution

```
#include<iostream>

using namespace std;

int main()

{

enum Fruits{apple,orange,guava,pinapple};

Fruits myfruit; //optional

int i;

cout<<"Enter your choice:";

cin>>i;
```

```
switch(i)
{
    case apple:
        cout<<"Your first fruit is Apple."; break;
    case orange:
        cout<<"Your second fruit is Orange."; break;
    case guava:
        cout<<"Your third fruit is Guava.";
        break;
    case pineapple:
        cout<<"Your forth fruit is Pineapples."; break;

}
return 0;
}
```

```
C:\Users\Asus\Desktop\C++Programes\ENUM_Pro.exe  
Enter your choice:3  
Your forth fruit is Pinapple.  
Process returned 0 (0x0)   execution time : 2.656 s  
Press any key to continue.  
-
```

Inventing a Boolean type

- There is no Boolean (false or true) datatype in C++; the integer values 0 and 1 are used instead.
- However, it is easy to invent your own Boolean type using enum.

```
enum boolean {false,true}; //false=0,true=1
```

Specifying integer values

- In enum specifier, the first name was given by the integer value 0, the second the value is 1, and so on.
- This ordering can be altered by using an equals sign to specify a starting point other than 0.

```
enum suit { clubs=1,diamonds,hearts,spades};
```

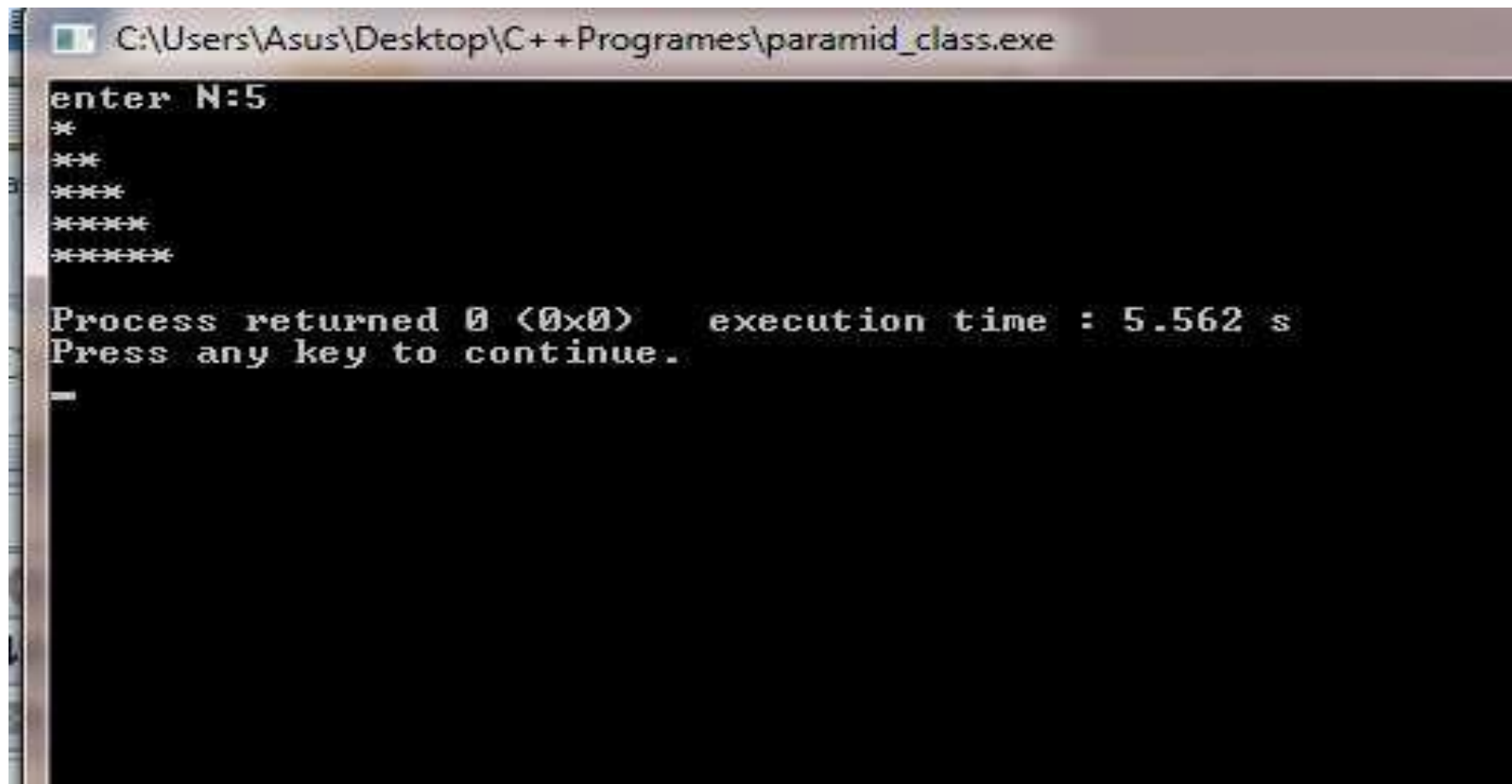
Class

- The class type enables us to create sophisticated use defined type.
- We provide data items for the class and the operation that can be performed on the data.
- Syntax:

```
class class_name  
{  
data member variables;  
data member methods/functions;  
};
```

Exercise

- Draw the below mentioned pyramid using the concept of class



```
C:\Users\Asus\Desktop\C++Programes\paramid_class.exe
enter N:5
*
**
***
****
*****

Process returned 0 (0x0)   execution time : 5.562 s
Press any key to continue.
_
```

Solution

```
#include<iostream>
using namespace std;
```

```
class Pyramid
{
int i,n,j;
public:
void getdata();
void buildpyramid();

};

void pyramid::getdata()
{
cout<<"enter N:"; cin>>n;
}
```



```

void piramid::buildpyramid()
{
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=i;j++)
        {
            cout<<"*";
        }
        cout<<"\n";
    }
}

int main()
{
    Pyramid p;
    p.getdata();
    p.buildpyramid();
    return 0;
}

```

```
C:\Users\Asus\Desktop\C++Programes\paramid_class.exe
enter N:5
*
**
***
****
*****

Process returned 0 (0x0)   execution time : 5.562 s
Press any key to continue.
_
```

Derived Data Type

- The derived data type are:
 - Array
 - Function
 - pointer

Array

- An array is a series of elements of the same data type placed in contiguous memory location.
- Like regular variable ,an array must be declare before it is used.
- Syntax:

`data_type name[size];`

- Ex:

`char name[10];`

`int student[10];`

Function

- A function is a group of statement that together perform a task.
- Every c++ program has at least one function that is `main()` and programmer can define additional function.
- You must define function prototype before use them.
- A function prototype tells compiler the name of the function ,return type, and parameters.

- Syntax:

`return_type function_name(parameter_list);`

- Ex:

`int max(int n1,int n2);`

Exercise

- Write a program to demonstrate the working of function in C++ to find out the maximum of 2 numbers entered by the user

Solution

```
#include<iostream> using namespace std;
```

```
int max(int n1,int n2);
```

//declaration

```
int main()
```

```
{
```

```
int n1; int n2; int a;
```

```
cout<<"Enter Number1: "<<"\n";
```

```
cin>>n1;
```

```
cout<<"Enter Number2: "<<"\n";
```

```
cin>>n2;
```

```
a=max(n1,n2);
```

//calling function

```
cout<<"max value is "<<a<<"\n";
```

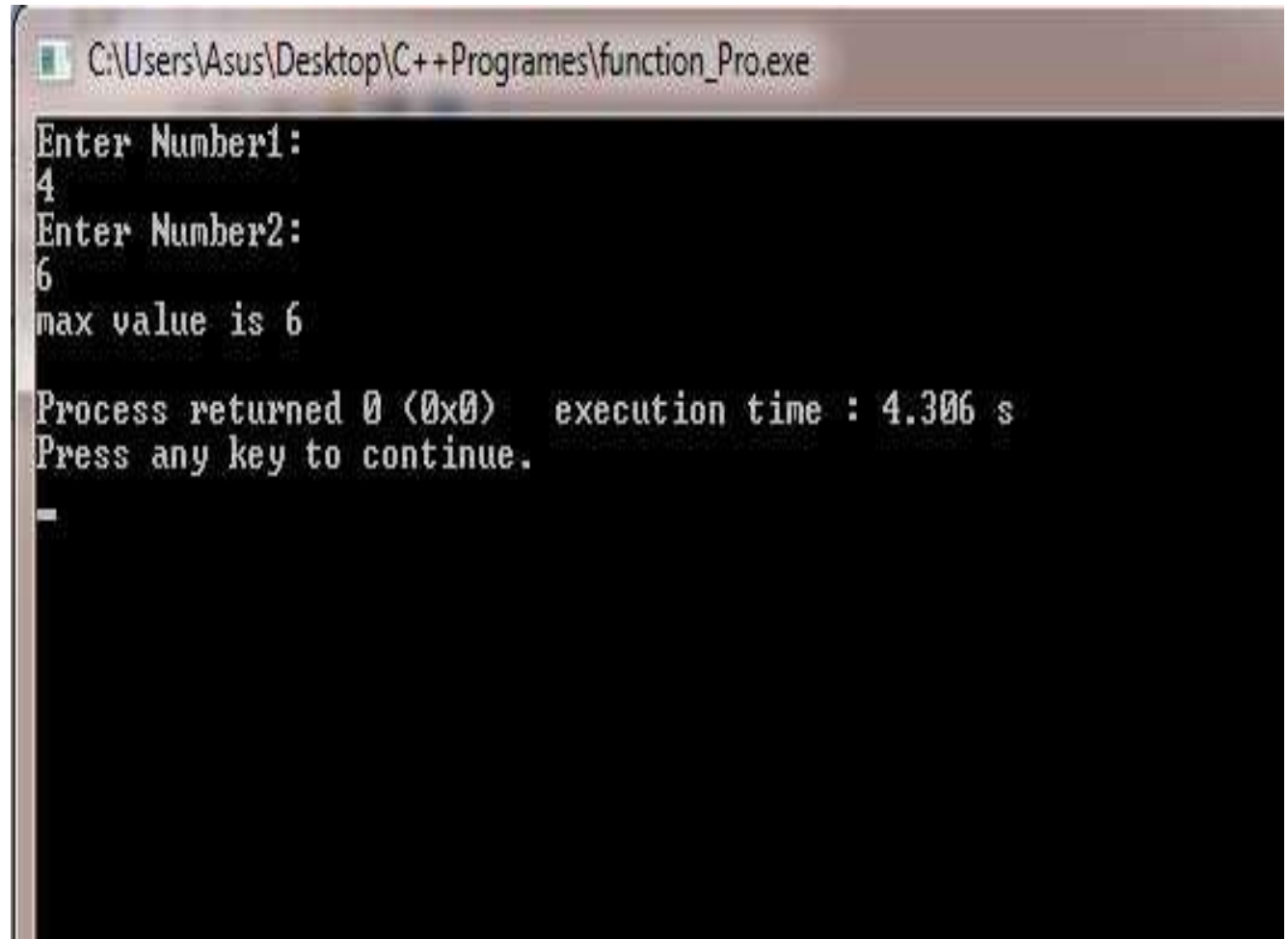
```
return 0;
```

```
}
```



```
int max(int n1,int n2)
{
    int result;
    if(n1>n2)
        result=n1;
    else
        result=n2;

    return result;
}
```



```
C:\Users\Asus\Desktop\C++Programes\function_Pro.exe
Enter Number1:
4
Enter Number2:
6
max value is 6
Process returned 0 (0x0)   execution time : 4.306 s
Press any key to continue.
_
```

Pointer

- Pointer is basically the same as any other variable, difference about them is that instead of containing actual data they contain a pointer to the memory location where information can be found.
- Basically, pointer is variable whose value is address of another variable.
- Syntax:

`type *var_name;`

- Ex:

`int *p;`

- There are few operation which will do frequently with pointer is:
 1. We define a pointer variable.
 2. Assign address of variable to a pointer and
 3. Finally access the value at the address available in the pointer variable.
- Like,
 1. `int var=20;`
 2. `p =&var;`
 3. `int *p;`

Pointer Program

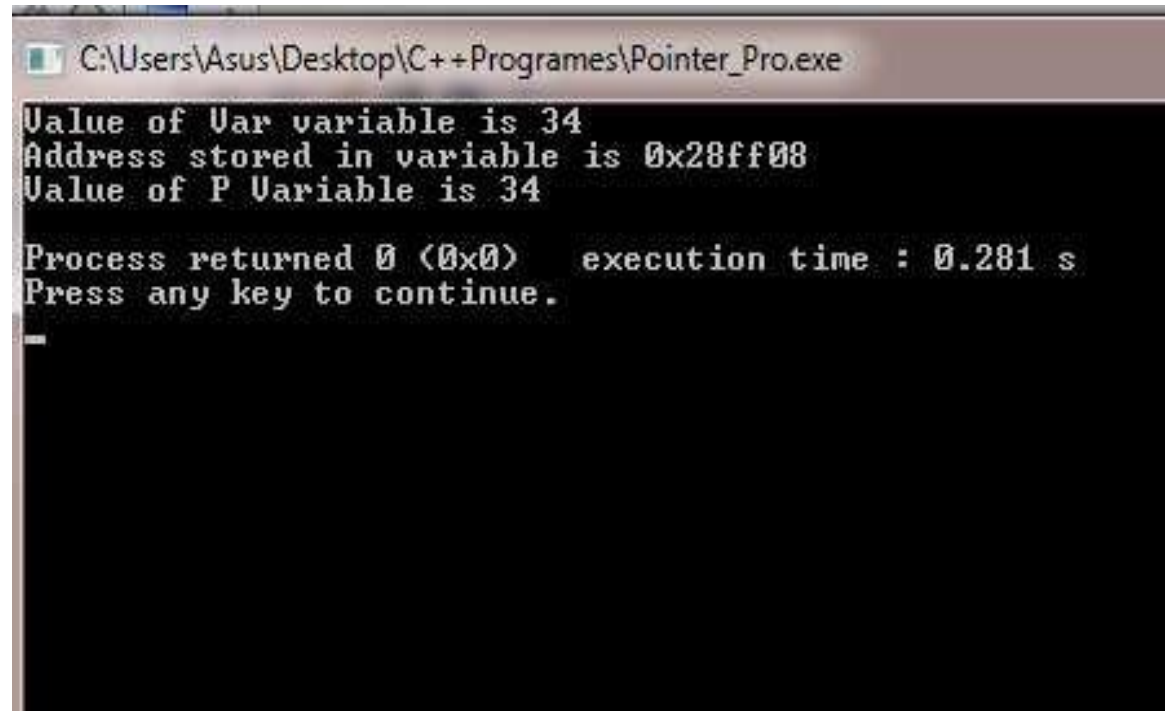
```
#include<iostream>
using namespace std;
```

```
int main()
{
int var=34; int *p;
p=&var;
```

```
cout<<"Value of Var variable is
"<<var<<"\n";
```

```
cout<<"Address stored in
variable is "<<p<<"\n";
```

```
cout<<"Value of P Variable is
"<<*p<<"\n"; return 0;
}
```



```
C:\Users\Asus\Desktop\C++Programes\Pointer_Pro.exe
Value of Var variable is 34
Address stored in variable is 0x28ff08
Value of P Variable is 34

Process returned 0 (0x0)   execution time : 0.281 s
Press any key to continue.
-
```

Symbolic Constant Type

- Constant is an identifier with associated value which can not be altered by the program during execution.
- You must initialize a constant when you create it, you can not assign new value later after constant is initialized.
- Symbolic constant is a constant that is represented by name.
 - **Defining constant with ' #define'.**
 - **Defining constant with 'const'.**
 - **Defining constant with 'enum'.**

- Ex:

```
#define pi 3.14
```

```
const max=10;
```

```
enum char{a,b,c};
```

Type Compatibility

- The sizeof is a keyword but it is compile time operator that determine the size, in byte, of a variable or data type.
- It can be used to get the size of classes, structure, union and any other user defined data type.

- Syntax:

`sizeof(data_type)`

- Ex:

`sizeof(x) or sizeof(int)`

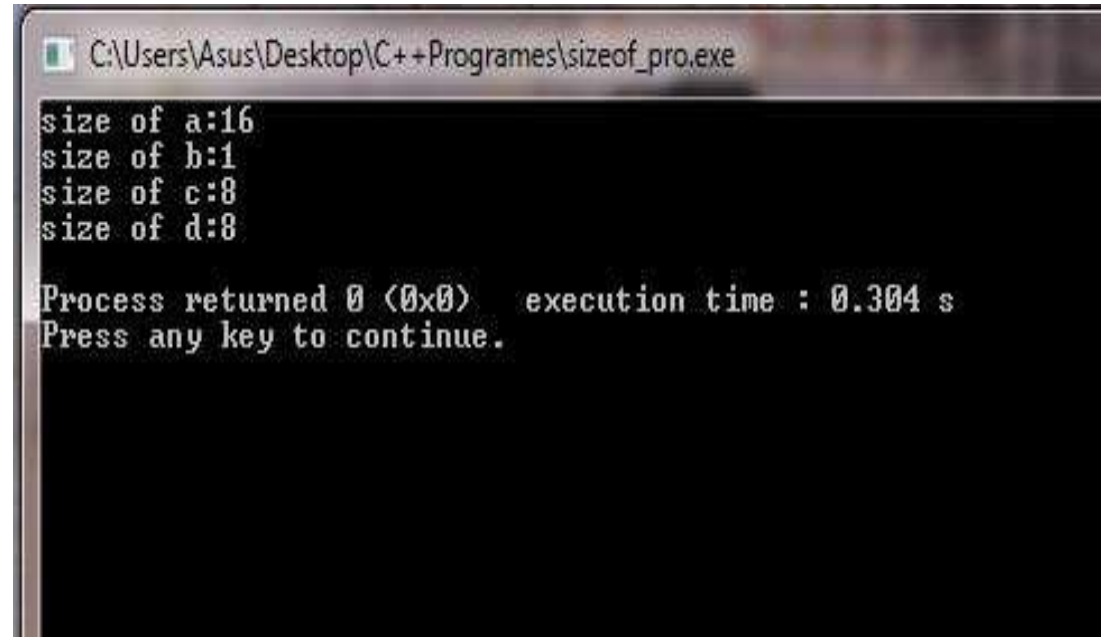
- sizeof operater will return integer value.

Program: Use Of sizeof Operator

```
#include<iostream>
using namespace std;

int main()
{
int a,x; char b;
float c=10.02; double d;
```

```
cout<<"size of a:"<<sizeof(a)*sizeof(x)<<"\n"; \*nested sizeof*\
cout<<"size of b:"<<sizeof(b)<<"\n";
cout<<"size of c:"<<sizeof(10.02)<<"\n";
cout<<"size of d:"<<sizeof(d)<<"\n";
return 0;
}
```



The screenshot shows a Windows command prompt window titled "C:\Users\Asus\Desktop\C++Programes\sizeof_pro.exe". The output of the program is displayed as follows:

```
size of a:16
size of b:1
size of c:8
size of d:8

Process returned 0 (0x0)   execution time : 0.304 s
Press any key to continue.
```


Variable In C++

- Variable is a place to store information.
- A variable is a location in your computer's memory in which you can store a value and from which you can later retrieve the value.
- This is temporary storage, when you turn the computer off these variable are lost.
- Like c, in C++ all variable must be declare before we used it.
- The difference is that in c, it requires to be defined at the beginning of a scope, rather C++ allow you to declaration of variable anywhere in the scope.

- This means that a variable can be declared right at the place of its first use.
- Declaration Of variable: `char name[12];`

`int a;`

`float num;`

- If you are going to declare more than one variable having same data type, you can declare all of them in single statement by separating their identifiers with commas.
- For ex:

`int a,b,c;`

- At the other-end you can assign value to the variable when you declare them its called initialization.

- Ex:

```
int a=0;
```

```
char name[]={'h','e','l','l','o'};
```

Dynamic initialization:

The additional feature of c++ is ,to initialize the variable at run time it is called dynamic initialization.

For example:

```
int c= a+b;
```

```
float avg=sum/n;
```

Reference variable:

- C++ references allow you to create a second name for the variable that you can use to read or modify the original data stored in that variable.
- This means that when you declare a reference and assign it a variable it will allow you to treat the reference exactly as though it were the original variable for the purpose of accessing and modifying the value of the original even if the second name(reference) is located within the different scope.
- Variable with different name share same data space.

- Syntax:

```
data_type& ref_name= var_name;
```

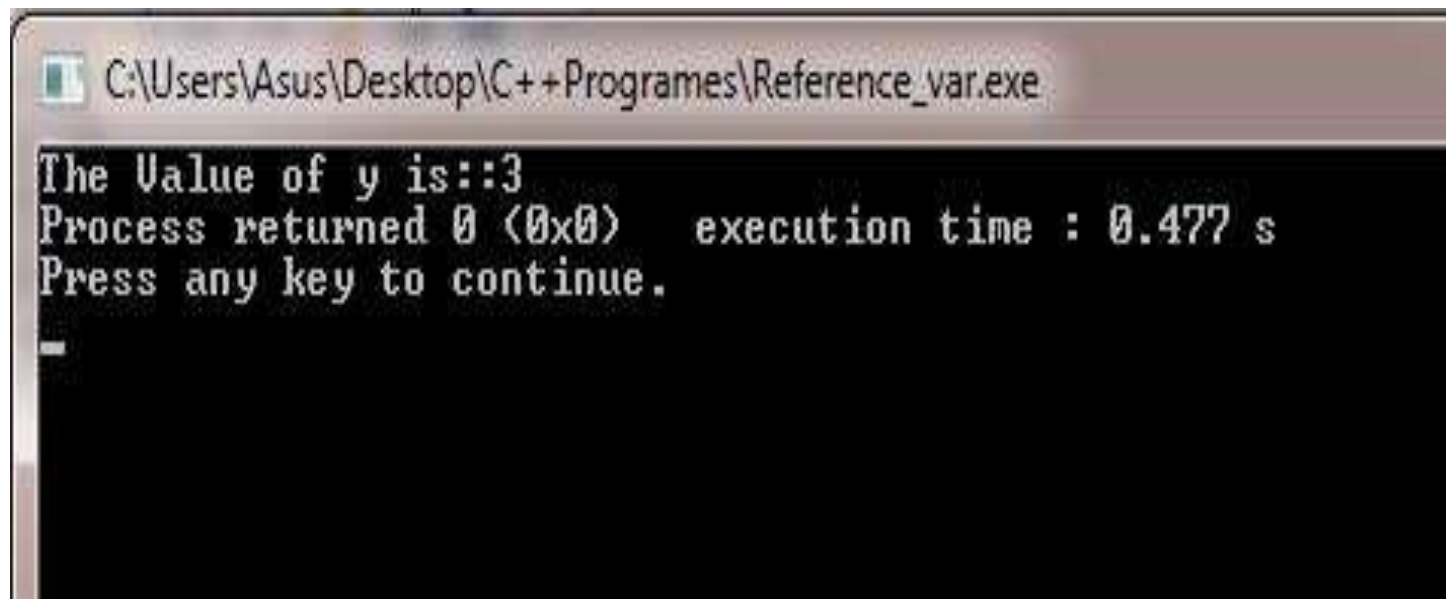
- Ex:

```
int a;
```

```
int& i=a;
```

Reference Program

```
#include<iostream> using namespace std; int  
main()  
{  
int x[5]={1,2,3,4,5};  
int& y=x[2];  
cout<< "The Value of y is::" << y; return 0;  
}
```



```
C:\Users\Asus\Desktop\C++Programes\Reference_var.exe  
The Value of y is::3  
Process returned 0 (0x0) execution time : 0.477 s  
Press any key to continue.  
-
```

Operators In C++

All the operator in c is available in c++,but some additional operator that are not in c are:

- << Insertion operator
- >> Extraction operator
- :: Scope resolution operator
- * Pointer to member operator
- delete** memory release operator
- endl** line feed operator
- new** memory allocation operator
- setw** field width operator

Scope Resolution Operator(::)

- Basically variable are two types ,local variable and global variable.
- The variable which are declared within a block or function, that are used only inside a function is called local variable.
- The global variable are declared out side function and used anywhere in the program.

- A situation occurs when both global and local variable name are same and we have to use global variable inside a local block or function, then it is only possible by using Scope Resolution Operator, it is used with global variable.
- Syntax:

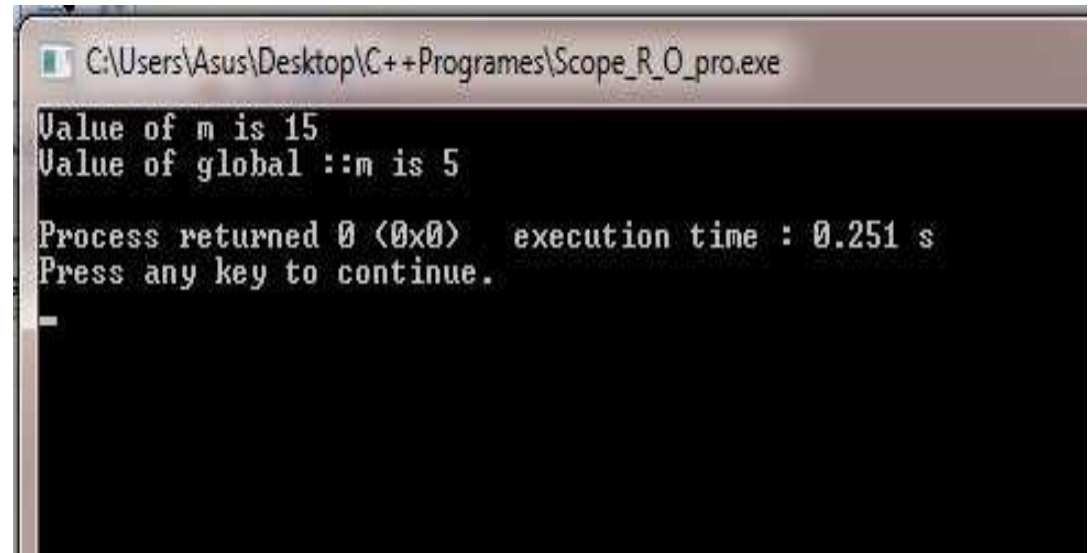
 `::variable_name;`
- A major application of the scope resolution operator is in the classes to identify the class to which a member function belongs.

Program Using :: Operator

```
#include<iostream>
using namespace std;

int m=5; //global
          declaration

int main()
{
    int m=15;
    cout<<"Value of m is "<<m<<"\n";
    cout<<"Value of global ::m is "<<::m<<"\n";
    return 0;
}
```



The screenshot shows a Windows command prompt window titled "C:\Users\Asus\Desktop\C++Programes\Scope_R_O_pro.exe". The output of the program is displayed as follows:

```
Value of m is 15
Value of global ::m is 5
Process returned 0 (0x0)   execution time : 0.251 s
Press any key to continue.
```

Member Dereferencing Operator

- The member dereferencing operator are used in class, a class contain various type of data and function as member.
- The dereference operator is also known as the indirection operator.
- A c++ also permit us to access the class member through pointer, so required a pointer to member operator which are,
 - `::*` to declare a pointer to a member class
 - `.*` to access a member using object name or Pointer to that member.
 - `->*` to access a member using a pointer to the object and a pointer to that member.

The Dereference Operator (*)

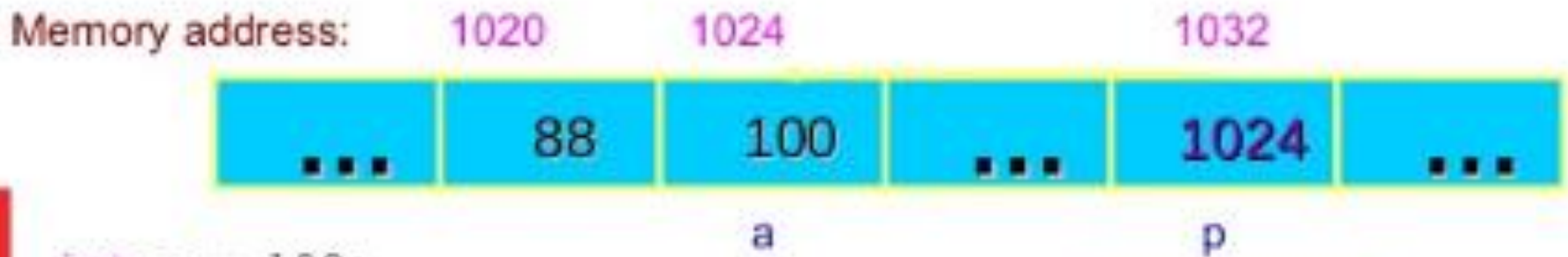
```
#include <iostream>
using namespace std;
int main()
{
    int a=5;
    cout<<a<<endl;
    cout<<&a<<endl;
    cout<<*&a;

    return 0;
}
```

```
5
0027FEA0
5
```

Dereferencing Operator *

- We can access to the value stored in the variable pointed to by using the dereferencing operator (*),



```
int a = 100;
int *p = &a;
cout << a << endl;
cout << &a << endl;
cout << p << " " << *p << endl;
cout << &p << endl;
```

Result is:

100
1024
1024 100
1032

Memory Management Operators

- C++ using two unary operators to new and delete , that perform the task of allocating and freeing the memory in better and easy way.
- A new operator can be create object of any type.

- Syntax:

```
pointer_variable= new data_type;
```

- Ex:

```
int *p=new int;
```

- A delete operator deallocate pointer variable.

- Syntax:

```
delete pointer_variable;
```

- Ex:

```
delete p;
```

- You can also create space for any data type including user-defined data type like array, structure, classes ,etc.
- Syntax :

```
pointer_variable=new data_type[size];
```

- Ex:

```
int *p=new int[100];
```

- Delete array element.
- Syntax:

```
delete [] pointer_variable;
```

- Ex:

```
delete []p;
```


Program Using New And Delete Operator

```
#include<iostream> using namespace std; int main()
```

```
{
```

```
int *a=new int; int *b =new int;
```

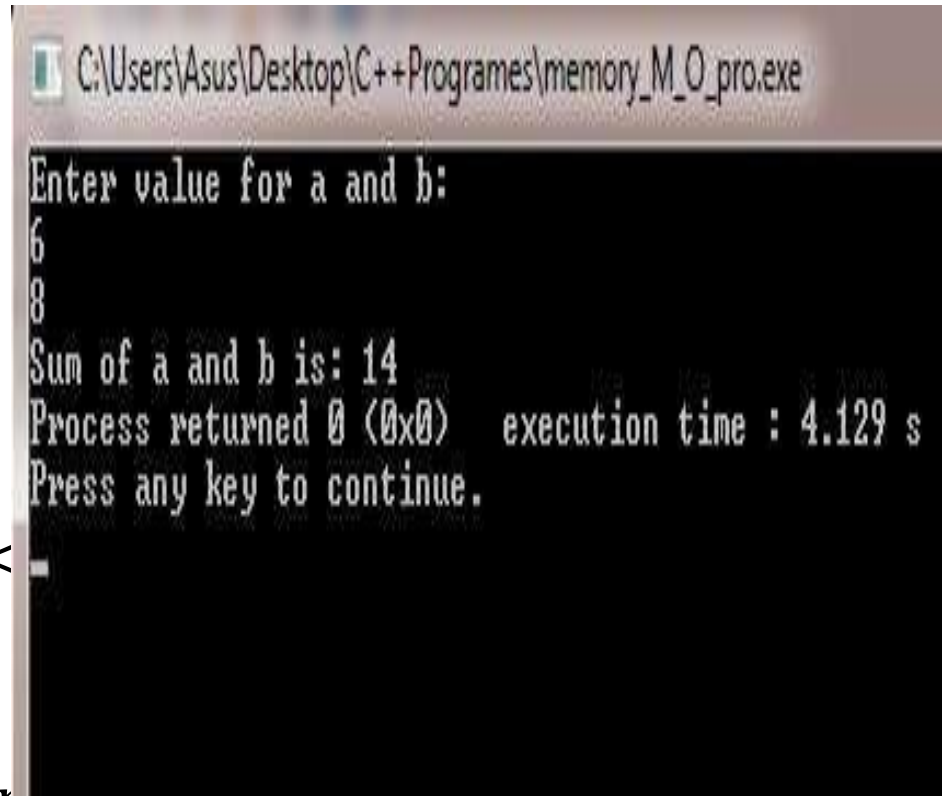
```
int *sum= new int;
```

```
cout<<"Enter value for a and b:"<
```

```
*sum=*a+*b;
```

```
cout<<"Sum of a and b is: "<<*sum, delete a,
```

```
delete b; return 0;
```



```
C:\Users\Asus\Desktop\C++Programes\memory_M_O_pro.exe
Enter value for a and b:
6
8
Sum of a and b is: 14
Process returned 0 (0x0) execution time : 4.129 s
Press any key to continue.
```

Manipulators

- C++ provides the operator called manipulator for formatting the output or manipulate the data in the way programmers wants to display it.
- Most commonly used manipulators are:
 - endl
 - setw()
- endl: it causes a linefeed to be inserted into the stream. It has the same effect as sending a single new line(“\n”) character.
- setw() is used to provide the field width for the value and display them right-justified.

Input manipulators

ws	Extract whitespaces (function)
-----------	---------------------------------

Output manipulators

endl	Insert newline and flush (function)
ends	Insert null character (function)
flush	Flush stream buffer (function)

Parameterized manipulators

These functions take parameters when used as manipulators. They require the explicit inclusion of the header file `<iomanip>`.

setiosflags	Set format flags (function)
resetiosflags	Reset format flags (function)
setbase	Set basefield flag (function)
setfill	Set fill character (function)
setprecision	Set decimal precision (function)
setw	Set field width (function)

- Like,
`int x=22;`
`cout<<setw(5)<<x;`

			2	2
--	--	--	---	---

- The `<iomanip>` header file provide rich set of manipulators.

```
// CPP Program to test std::setfill manipulator
#include <iostream>
#include <iomanip> // std::setfill, std::setw

int main()
{
    // setfill is x and width is set as 10
    std::cout << std::setfill('x') << std::setw(10);

    std::cout << 77 << std::endl;

    std::string str = "Geeks";

    // setfill is G and width is set as 10
    // And std::left is used set str to left side
    std::cout << std::left << std::setfill('G') << std::setw(10);

    std::cout << str << std::endl;
    return 0;
}
```

Output

xxxxxxxx77

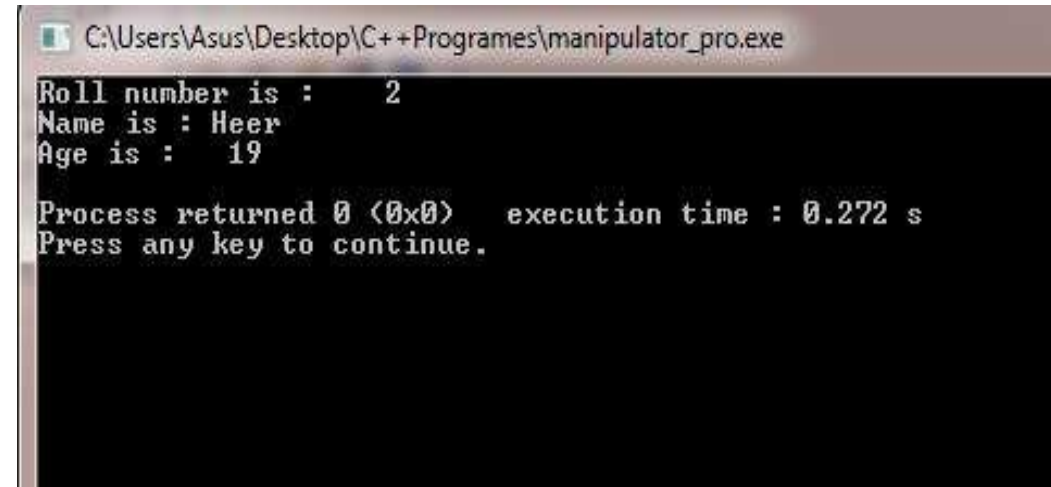
GeeksGGGGG

Manipulator Program

```
#include<iostream> #include<iomanip> using namespace  
std;
```

```
int main()  
{  
int roll_no=2;  
char name[]="Heer"; int age=19;
```

```
cout<<"Roll number is : "<<setw(5)<<roll_no<<endl;  
cout<<"Name is : "<<setw(5)<<name<<endl; cout<<"Age is  
:"<<setw(5)<<age<<endl;  
}
```



```
C:\Users\Asus\Desktop\C++Programes\manipulator_pro.exe  
Roll number is : 2  
Name is : Heer  
Age is : 19  
Process returned 0 (0x0) execution time : 0.272 s  
Press any key to continue.
```

Type Conversion

- Categorized in

1. Automatic conversions:

- When two operands of different types are encountered in the same expression, the lower-type variable is converted to the type of higher-type variable.

2. Casts:

- This is specified by the programmer, as opposed to the automatic data conversions. This is sometimes called *coercion*; the data is coerced into becoming another type

Type Casting

- While evaluating an expression if both operands for an operator are not of same data type, then using implicit conversion rule c++ convert smaller type to wider type.
- For example, if two operands are respectively int and float type, then int is converted into a float and then operation is performed.
- Many times implicit conversion cause the problem and we do not get proper result.

- For that, like c, c++ also uses explicit type casting with the type cast operator in two different style as follow:

- Syntax:

`(type_name) expression;`

`type_name (expression);`

- Ex:

`average=(float) sum/n;`

`average=float(sum)/n;`

Expression And Their Types

- An expression is combination of operators, constants and variables arranged as per the rule of language.
- It may also include function calls which return values.
- An expression may consist of one or more operands and zero or more operators to produce a value.

- Expression may consist of following seven types:
 - Constant expressions
 - Integral expressions
 - Float expressions
 - Pointer expressions
 - Relational expressions
 - Logical expressions
 - Bitwise expressions

- **Constant expression:**

It consist of only constant value.

Ex: $20 + 5 / 20$;

- **Integral expression:**

Integral expressions are those which produce integer results after implementing all the automatic and implicit type conversions.

Ex: $m * n - 5$; //m and n are integer

$5 + \text{int}(2.0)$;

- **Float expression:**

Floating expression are those which, after all conversions, produce floating point results.

Ex: $5 + \text{float}(10)$;

$x * y / 10$; //x and y are floating point

- **Pointer expression:**

Pointer expression are produce address values.

Ex: &m;

ptr; //m is variable and ptr is pointer

- **Relational expression:**

Relational expressions yield results of type bool which takes a value true or false.

Also known as boolean expression.

Ex: x <= y;

a+b = c+d;

If arithmetic expression are used either side of relational operator, they will be evaluated first then the results compared.

- **Logical expression**

Logical expression combine two or more relational expression and produces bool type result.

Ex: `a>b && x==10;`

`x==10 || y==5;`

- **Bitwise expression**

Bitwise expressions are used to manipulate data at bit level. They are basically used for testing or shifting bits.

Ex: `x<<3` `// shift three bit position to left`
 `y>>1` `// shift one bit position to right`

Special Assignment Expressions

- **Chained assignment:**

`x = (y=10);` or `x = y = 10;`

In above expression the 10 value is first assign to y and then to x.

Chained expression is not used to initialization at the time of declaration .

Like ,

`float a=b=12.34;` is wrong, instead of this we may write , `float a = 12.34, b= 12.34;` is correct.

- **Embedded assignment:**

$x = (y = 50) + 10;$

where $y=50$ is an assignment expression known as embedded assignment.

here, value 50 is assigned to y and then the result ($50 + 10 = 60$) is assigned to x .

like, $y=50;$

$x = y + 10;$

- **Compound assignment:**

Compound assignment operator ($+=$) is a combination of the assignment operator with a binary operator.

Ex: $x=x+10$ may be written as , $x +=10;$

Implicit Conversion

- Whenever data type are mixed in expression, c++ perform conversions automatically. This process is known as implicit conversion.
- If the operand type differ than the compiler converts one of them to match with the other using the rule that the “smaller” type is converted to the “wider” type.
- For example, if one operand is an int and other is float , the int is converted into float because float is wider than int.
- Whenever a char and short int appears in expression it is converted to an int.

Precedence & Associativity

Precedence is used to determine the order in which different operators in a complex expression are evaluated. Associativity is used to determine the order in which operators with the same precedence are evaluated in a complex expression.

precedence	operators	description	associativity
1	::	Scope resolution operator	Left-to-right
2	++ -- type() type{ } () [] . ->	Suffix/postfix increment decrement Function style Type cast Function call Array subscripting Element selection by Reference Element selection through pointer	

precedence	operators	description	associativity
3	<p>++ --</p> <p>+ -</p> <p>! ~</p> <p>(type)</p> <p>*</p> <p>&</p> <p>sizeof</p> <p>new, new[]</p> <p>delete ,</p> <p>delete[]</p>	<p>Prefix increment and decrement</p> <p>Unary plus and minus</p> <p>logical NOT and bitwise NOT</p> <p>C-style type cast</p> <p>Indirection (dereference)</p> <p>Address –of</p> <p>Sizeof</p> <p>Dynamic memory allocation</p> <p>Dynamic memory deallocation</p>	Right-to-left

precedence	operators	description	associativity
4	. * ->*	Pointer to member	Left-to-right
5	* / %	Multiplication, division and remainder	
6	+ -	Addition and subtraction	
7	<< >>	Bitwise left shift and right shift	
8	< <= > >=	Relational operator	
9	== !=	Relational is equal to and not equal to	
10	&	Bitwise AND	
11	^	Bitwise(exclusive)OR	
12		Bitwise (inclusive) OR	
13	&&	Logical AND	
14		Logical OR	

precedence	operators	description	associativity
15	?: throw = += -= *= /= %= <<= >>= &= ^= =	Ternary operator Throw operator Direct assignment operator Assignment by sum and difference Assignment by product quotient, and remainder Assignment by left shift and right shift Assignment by bitwise AND XOR and OR	Right-to-left
16	,	comma	Left-to-right

Category	Operator	Associativity
Postfix	<code>() [] -> . ++ --</code>	Left to right
Unary	<code>+ - ! ~ ++ -- (type) * & sizeof</code>	Right to left
Multiplicative	<code>* / %</code>	Left to right
Additive	<code>+ -</code>	Left to right
Shift	<code><< >></code>	Left to right
Relational	<code>< <= > >=</code>	Left to right
Equality	<code>== !=</code>	Left to right
Bitwise AND	<code>&</code>	Left to right
Bitwise XOR	<code>^</code>	Left to right
Bitwise OR	<code> </code>	Left to right
Logical AND	<code>&&</code>	Left to right
Logical OR	<code> </code>	Left to right
Conditional	<code>?:</code>	Right to left
Assignment	<code>= += -= *= /= %= >>= <<= &= ^= =</code>	Right to left
Comma	<code>,</code>	Left to right

Operator Precedence and Associativity

Highest



Lowest

Operator	Associativity
()	Left to right
++ -- ! + - (unary)	Right to left
* / %	Left to right
+ -	Left to right
< <= > >=	Left to right
== !=	Left to right
&&	Left to right
	Left to right
?:	Right to left
= += -= *= /=	Right to left

Control Structure

- Like c ,c++ also provide same control structure as follows:
 - if statement
 - if-else statement
 - switch statement
 - do-while statement
 - while statement
 - for statement

If statement:

syntax:

```
if (expression is true)
{
    action-1;
}

else
{
    action-2;
}
```

```
if(a>b)
{
    cout<<"a is max";
}
else
{
    cout<<"b is max";
}
```

Switch statement:

```
switch(expression)
{
case1:
action1; break;
case2:
action2;
break;
... default:
actionN;
}
```

```
switch(character)
{
case 'm':
cout<<"male candidate";
break;
case 'f':
cout<<"female candidate";
break; default:
cout<<"invalid gender code";
}
```

While statement:

syntax:

```
while(condition is true)
{
    action1;
}
```

```
while(i>5)
{
cout<<"number of class:"; i++;
}
```

Do-while statement:

syntax: do

{

action1;

}

while(condition is true);

```
i=0;  
do  
{  
cout<<i<<"\n"; i++;  
}  
while(i<5);
```

For statement:

syntax: for(initial_value;
 condition;
 increment/decrement)

```
{  
action1;  
}
```

```
for(int i=0;i<=5;i++)  
{  
  cout<<i<<"\n";  
}
```

Other Control Statements

1. The break statement
2. The continue statement
3. The goto statement

Review Questions

Answer the following questions :

1. Describe data types in C++ in Details.
2. Differentiate entry-controlled loop and exit-controlled loop.

Multiple Choice Questions

1. C++ statements that allow programmers to specifies that next statement to be executed may be other than next one in a sequence, is called as
 - A. Transfer of control
 - B. Transfer of statements
 - C. Transfer of program
2. C++ control structures are combined in
 - A. Control-structure stacking
 - B. Control-structure nesting
 - C. Control-structure flow
 - D. Both A and B
3. Which one is not a valid identifier?
 - A. rdd2
 - B. x (5)
 - C. _DATE_
 - D. A30
4. In C++ a name that is assigned by a user for a program elements such as variable, type, class, function or namespaces is called
 - A.Operators
 - B.Identifier
 - C.Literals
 - D.All of These

MCQ Answer

1. A

2. D

3. B

4. B